ALEA: A Fine-Grain Energy Profiling Tool

Mukhanov, L., Petoumenos, P., Wang, Z., Parasyris, N., Nikolopoulos, D. S., De Supinski, B. R., & Leather, H. (2017). ALEA: A Fine-Grain Energy Profiling Tool. *ACM Transactions on Architecture and Code Optimization* , *14*(1), [1]. https://doi.org/10.1145/3050436

Published in:
ACM Transactions on Architecture and Code Optimization

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

ALEA: A Fine-grained Energy Profiling Tool

Lev Mukhanov *Queen's University of Belfast*
Pavlos Petoumenos *University of Edinburgh*
Zheng Wang *Lancaster University*
Nikos Parasyris *Queen's University of Belfast*
Dimitrios S. Nikolopoulos *Queen's University of Belfast*
Bronis R. de Supinski *Queen's University of Belfast*
Hugh Leather *University of Edinburgh*

KEY WORDS:Energy profiling, sampling, energy efficiency, power measurement, ALEA

1 Introduction

In an era dominated by battery-powered mobile devices and servers hitting the power wall, increasing the energy efficiency of our computing systems is of paramount importance. While performance optimization is a familiar topic for developers, few are even aware of the effects that source code changes have on the energy profiles of their programs. Therefore, developers require tools that analyze the power and energy consumption at the source-code level of their programs.

Prior energy accounting tools can be broadly classified into two categories: tools that directly measure energy using on-board sensors or external instruments [1, 2, 3, 4, 5, 6]; and tools that model energy based on activity vectors derived from hardware performance counters, kernel event counters, finite state machines, or instruction counters in microbenchmarks [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. All of these tools can associate energy measurements with software contexts via manual instrumentation, context tracing, or profiling. However, these approaches have their limitations.

Direct power measurement: Using direct power measurement instruments, tools can accurately measure hardware-component-level and system-wide energy consumption. State-of-the-art external instruments such as the Monsoon power meter [18], direct energy measurement and profiling tools [3, 2], and internal energy and power sensors such as Intel's Running Average Power Limit (RAPL) [19] or on-board sensors [20] sample at a rate between 1 kHz and 3.57 kHz. This rate is unsuitable for energy accounting of individual basic blocks, functions, or even longer code segments that typically run for short time intervals. Thus, we cannot characterize the power usage of these fine-grained code units and will miss many optimization opportunities that can only be discovered through fine-grained energy profiling.

Activity-vector-based measurement: Tools that model energy consumption from activity vectors extracted from hardware performance counters can produce estimates for fine-grained code units but suffer from several shortcomings. Their accuracy is architecture- and workload-dependent [11, 9, 10, 17]. Thus, they are only suitable for a limited set of platforms and applications. Further, targeting a new platform or workload requires expensive benchmarking and training to calibrate the model. Finally, as we show later, these tools are not accurate enough to capture the power variation caused by the execution of different code regions.

To overcome the limitations of prior work, this paper presents ALEA, a new probabilistic approach for estimating power and energy consumption at a fine-grained level. Specifically, we propose a systematic *constant-power probabilistic model* (CPM) to estimate the power consumption of coarse-grained code regions with latency greater than the minimum feasible power-sensing interval and an improved

variation-aware probabilistic model (VPM) that estimates power more accurately by identifying high - frequency variations at the fine-grained level through additional profiling runs. CPM approximates the power consumption of a region under the assumption that power is constant between two successive readings of the system's power sensors. Its accuracy depends on the minimum interval that is allowed between two such readings. VPM relaxes this assumption and identifies power variation over intervals significantly shorter than the minimum feasible power-sensing interval. We validate CPM by comparing the results of profiling with real energy measurements on two different architectures using a set of diverse benchmarks. The experimental results show that CPM is highly accurate, with an average error below 2% for coarse-grained regions. However, for fine-grained regions this model can produce much higher errors. VPM successfully handles these cases, allowing energy accounting at a frequency that is up to two orders of magnitude higher than on-board power sensing instruments, with the error of its estimates at 6%, which is 92% lower than those of CPM.

Finally, we present two use cases of ALEA: fine-grain memoization in financial option pricing kernels to optimize performance under power caps; and fine-grained DVFS to optimize energy consumption in a sequence alignment algorithm. These use cases show that our approach can profile large real-world software systems to understand the effect of optimizations on energy and power consumption.

The contributions of this paper are:

- A new method for accurate energy accounting in code blocks with finer granularity than that of power-sensing instruments on real hardware;
- Two first-of-their-kind, portable probabilistic models for accurate energy accounting of code fragments of granularity down to 10 μSec, or two orders of magnitude finer than direct power measurement instruments;
- Low-latency and non-intrusive implementation methods for fine-grained, online energy accounting in sequential and parallel applications;
- Tangible use cases that demonstrate energy efficiency improvements in realistic applications, including energy reduction by 2.87$\times$ for an industrial strength option pricing code executed under a power cap and up to 9% reduction in energy consumption in a sequence alignment algorithm.

The rest of this paper is structured as follows. Section 2 provides a short overview of our profiling approach. Section 3 details our energy sampling and profiling models, while Section 4 examines the key aspects of their implementation. Section 5 demonstrates the accuracy of the proposed probabilistic models. Section 6 details opportunities and limitations of ALEA, including the effect of hysteresis on physical power measurements and an analysis of ALEA's accuracy compared against power models based on hardware performance counters. Section 7 presents how ALEA can be used to improve energy efficiency in realistic applications. Section 8 discusses previous work in this area and Section 9 summarizes our findings.

2 Overview of ALEA

ALEA provides highly accurate energy accounting information at a fine granularity to reason about the energy consumption of the program. It collects power and performance data at runtime without modifying the binary. ALEA uses probabilistic models based on its profiling data to estimate the energy consumption of code blocks.

2.1 Definitions

We define terms that we use throughout the paper as follows:

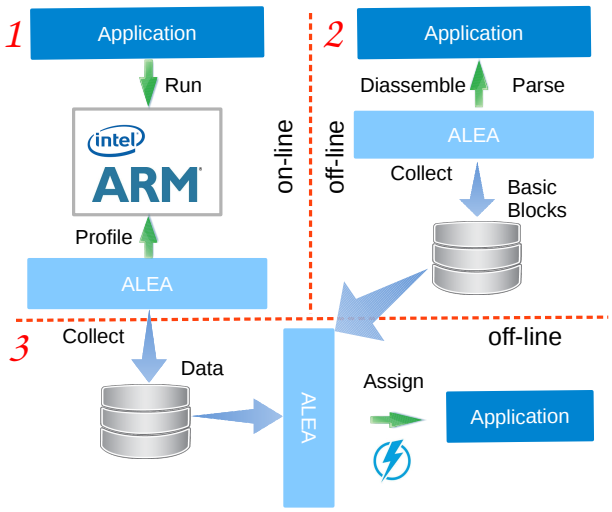

Figure 1: Workflow of ALEA

- **Code block:** a region of code that the user defines at the final profiling stage and that can include one or more basic blocks or procedures;
- **Sampling interval:** the time between two consecutive samples;
- **Power-sensing interval:** the interval during which a single power measurement is taken in a computing system;
- **Fine-grained block:** a code block that has an average latency less than the power-sensing interval;
- **Coarse-grained block:** a code block that has an average latency of at least the power-sensing interval.

2.2 Workflow

Figure 1 illustrates the three steps of the ALEA workflow:

Profiling. The instrumented program is profiled using several representative input data sets. During each run, ALEA periodically samples instruction addresses and records readings from a power measurement instrument such as an on-board energy sensor or a software-defined power model (e.g., based on hardware counters).

Disassembling and parsing. After each run, ALEA disassembles the binary and identifies basic blocks in the code.

Energy accounting. As we show in Section 5, the default model of ALEA (CPM, see Section 3.1) provides accurate energy and power estimates for coarse-grained blocks.

To obtain accurate energy estimates for specific fine-grained code blocks, the user should choose to use our second model (VPM). VPM overcomes the coarse-granularity of the power sensors and provides accurate energy estimates over intervals with latency down to 10 μs. Finally, ALEA probabilistically estimates the execution time and power consumption of each code block.

3 Energy Accounting based on a probabilistic model

This section details our probabilistic energy accounting models. We first present our execution time profiling method in Section 3.1, followed by our sampling method in Section 3.2. We then describe our constant and variation-aware power models in Section 3.3. We extend our methodology to multi-threaded code in Section 3.4. We close this section with an analytic study of the accuracy of our energy accounting models.

3.1 Execution Time Profiling Model

Our execution time profiler randomly samples the program counter multiple times per run. Based on the total execution time of the program and the ratio of samples of a code block to the total number of samples, we estimate the time that a block executes. More formally, we define a processor clock cycle as a unit of the finite population (U). It has been demonstrated that the execution time of code block blm can be estimated based on the probability to sample this block at a random cycle [21, 22]:

$$\hat{t}_{blm} = \hat{p}_{blm} \cdot t_{exec} = \frac{n_{blm} \cdot t_{exec}}{n} \tag{1}$$

In Equation (1) t_{blm} is the the total execution time estimate of instances of blm, t_{exec} is the program's total execution time, $\hat{p}_{blm}$ is the probability to sample block blm, n_{blm} is the number of samples of any instruction from blm, and n is the total number of samples. Equation (1) captures that the probability of sampling a code block at a random clock cycle is the ratio of its execution time to the program's total execution time.

3.2 Systematic Sampling

We use systematic sampling, which approximates random sampling. It selects the first sample (unit) randomly from the bounded interval $[1, \, length\ of\ sampling\ interval]$ and other samples are selected from an ordered population with an identical sampling interval. We subsequently sample in the same run every $sampling\ interval$ clock cycles. The random selection of the first sample ensures that sampling is sufficiently random. Even if the sampling interval aligns with a pattern in the execution of the program, we can still obtain accurate estimates by aggregating multiple profiling runs, each one with a different initial sampling time.

3.3 Probabilistic Power Models

We apply the same probabilistic approach that we use for time profiling in order to profile power and energy. We consider power consumption as a random variable that follows a normal distribution and is a characteristic associated with the clock cycle population. We simultaneously sample the program counter and power consumption, which we assign to the sampled code block.

Assuming n_{blm} samples of block blm, we estimate its mean power consumption as:

$$\widehat{pow}_{blm} = \frac{1}{n_{blm}} \cdot \sum_{i=1}^{n_{blm}} pow^{i}_{blm} \tag{2}$$

where pow^{i}_{blm} is the power consumption associated with the ith sample of block blm.

We estimate the energy consumption of blm as:

$$\hat{e}_{blm} = \widehat{pow}_{blm} \cdot \hat{t}_{blm} \tag{3}$$

Our model can derive energy estimates for any system component (e.g. processors, DRAM, on-chip GPU) for which a hardware or software-defined power sensor exists. The rest of this paper considers

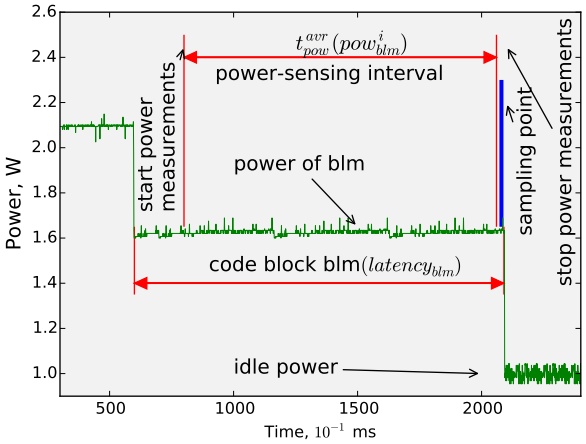

Figure 2: Power measurements for CPM

only processor energy accounting. The model implies that we measure the instantaneous power consumption of a sampled code block. However, power meters can only average measurements over a certain interval and power measurement can itself have a latency of hundreds of microseconds. We next discuss our two approaches to approximate the actual power consumption of the code block.

3.3.1 Constant Power Model

Figure 2 shows an example of the power measurement process of our Constant Power Model (CPM). When we sample the instruction pointer (the blue line), we interrupt the profiled threads, which causes the power consumption to drop quickly to the idle state, where it stays while all threads are paused. If we then measure power, this behavior would impact our measurement significantly. Thus, we measure power before sampling the current instruction pointer.

This model provides accurate power estimates for coarse-grained blocks. However, the power-sensing interval necessarily adjusts power consumption for other code blocks with potentially different power behavior. CPM, as well as all other existing energy profiling approaches, assumes that power remains constant throughout the power-sensing interval. If the power level varies during this interval, these energy profiling techniques do not capture that power variation.

3.3.2 Variation-Aware Power Model

Accurately measuring power for fine-grained code blocks requires a more sophisticated approach, which our *Variation-aware Power Model* (VPM) provides. We isolate each block's power consumption from that of the neighboring code blocks through multiple power measurements for the exact same sequences of code blocks. By pausing (or not) the execution of the last block in this sequence, we control whether it contributes (or not) to the power measurement. We subtract the power associated with a sequence that omits the block from that of one that includes it to obtain the power and energy consumption of that block.

The top plot in Figure 3 shows this process in more detail. VPM partially decouples measuring power from sampling. We initiate the power measurement at point `A` in the plot. The actual sample is taken after a certain amount of time, at point `B`. Sampling causes the program to pause, bringing the power consumption down to pow_{idle}. At the end of the power-sensing interval, point `C`, we obtain $pow^{no\_delay}_{A \to C}$.

Every time point `B` occurs at the beginning of an iteration of code block *blm*, the power reading is roughly the power consumption of the preceding blocks, excluding the contribution of *blm*. To obtain

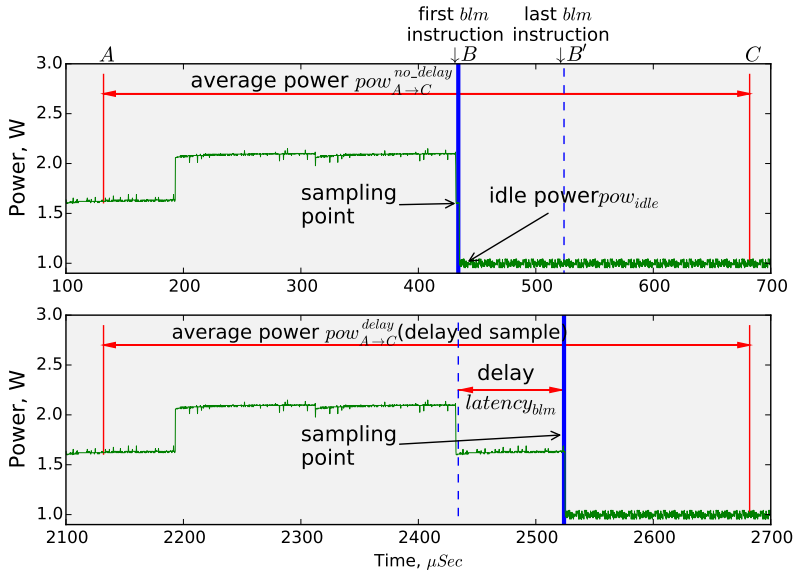

Figure 3: Power measurements for VPM

their power consumption *including* the contribution of *blm*, we delay sampling for an interval equal to *blm*'s latency for half of the samples.

The bottom half of Figure 3 shows that this delay causes half of the samples that would be taken at `B` to be taken at `B'` (*blm*'s end). The power-sensing interval covers the same preceding blocks as in the case of sampling at `B` but also includes *blm*. The difference of the two power readings gives us the energy and power consumption of *blm*:

$$e^i_{blm} = pow^{delay}_{A\to C} \cdot t^{avr}_{pow} - pow^{no\_delay}_{A\to C} \cdot t^{avr}_{pow} + pow_{idle} \cdot latency^i_{blm}. \tag{4}$$

$$pow^i_{blm} = \frac{pow^{delay}_{A\to C} - pow^{no\_delay}_{A\to C}}{latency^i_{blm}} \cdot t^{avr}_{pow} + pow_{idle}, \tag{5}$$

where $pow^{delay}_{A\to C}$ denotes the power measurements taken when we sample the last instruction of *blm* at `B'`, $pow^{no\_delay}_{A\to C}$ denotes power measurements taken when the first instruction of the code block was sampled at `B`, pow_{idle} is the idle power consumption obtained after interrupting the profiled threads and t^{avr}_{pow} is the power-sensing interval.

Our approach assumes that the power readings correspond to observations of a random variable with a normal distribution. Thus, we can average $pow^{delay}_{A\to C}$ and $pow^{no\_delay}_{A\to C}$ over all samples to estimate the power consumption for *blm* as:

$$\widehat{pow}_{blm} = \frac{\widehat{pow}^{delay}_{A\to C} - \widehat{pow}^{no\_delay}_{A\to C}}{\widehat{latency}_{blm}} \cdot t^{avr}_{pow} + pow_{idle}, \tag{6}$$

where $\widehat{latency}_{blm}$ is the average latency of the code block.

For VPM to produce accurate results, all power readings associated with the same sample, delayed or not, must cover the power consumption of the same code blocks. The time between the start of the power measurement and the sampling point must be constant, as must the time between the sampling point and the end of the measurement. We carefully implement VPM to meet these requirements as much as possible. The only source of error that we cannot control is variations of the execution path and the power consumption before the targeted code block. Given the regular nature of most applications,

Code blocks	Samples	Time	Energy	Power
core1:cb100,core2:cb100	163	21.99 +/-0.44	67.95 +2.71/-2.65	3.09 +/-0.06
core1:cb100,core2:sys	94	0.95 +/-0.19	1.66 +0.42/-0.39	1.75 +/-0.08
core1:cb161,core2:cb161	91	0.92 +/-0.18	2.29 +0.54/-0.51	2.48 +/-0.07

Figure 4: Profiling results for the `heartwall` benchmark on the Exynos platform

these variations are not a significant source of error. In Section 5.2 we validate VPM and we show that it accurately estimates power consumption.

If the latency of profiled code blocks is small enough then the accuracy of power sensors could be insufficient to evaluate the difference between $\widehat{pow}_{A\to C}^{delay}$ and $\widehat{pow}_{A\to C}^{no\_delay}$ (see Section 5.3). VPM may obtain erratic power measurements due to insufficient accuracy of the power sensors. To overcome this problem, we force upper and lower bounds on VPM power readings to correspond to the peak power consumption of the processor and pow_{idle} respectively.

3.4 Multi-threaded Code

Multi-threaded applications execute many code blocks in parallel on different cores. To profile applications running on the same processor package at the code block level, we define a vector of code blocks that execute in parallel on different cores as:

$$\overrightarrow{blm} = blm_{thr_1}, blm_{thr_2}, \ldots, blm_{thr_l}. \tag{7}$$

We use the same time and energy profiling models as in sequential applications. The only difference is that we associate multi-threading samples with vectors of code blocks. We distribute the execution time and energy across code block vectors as:

$$\hat{t}_{\overrightarrow{blm}} = \hat{p}_{\overrightarrow{blm}} \cdot t_{exec} = \frac{n_{\overrightarrow{blm}} \cdot t_{exec}}{n}, \tag{8}$$

$$\widehat{pow}_{\overrightarrow{blm}} = \frac{1}{n_{\overrightarrow{blm}}} \cdot \sum_{i=1}^{n_{\overrightarrow{blm}}} pow_{\overrightarrow{blm}}^{i}. \tag{9}$$

We examine all running threads collectively during sampling, because they share resources. Such shared resources include caches, buses, and network links, all of which can significantly increase power consumption under contention. We could apportion power between threads based on dynamic activity vectors that measure the occupancy of shared hardware resources per thread [8]. However, these vectors are difficult to collect and to verify as current hardware monitoring infrastructures do not distinguish between the activity of different threads on shared resources.

Figure 4 shows the results of profiling the `heartwall` benchmark (Rodinia suite) on the Exynos platform (see Section 4.1), including 95% confidence intervals for each estimate. `core1 :cb100, core2 :cb100` denotes the code block with identification `100` executed simultaneously on the first and second cores. Similarly, `cb161` is the code block with identification `161` and `sys` is a code block from the `pthread_cond_wait()` function of the `glibc` library; it executes a `mutex` lock that keeps a core in sleep mode until the corresponding (pthread_cond_signal) occurs. We see that `cb100` executed in parallel consumes almost twice as much power than when it is co-executed with the `sys` block. In the second case, the second core is sleeping and consumes almost no power. The difference in power consumption between parallel execution of `cb161` and `cb100` is related to the cache access intensity of these blocks. Our earlier work shows that dynamic power consumption grows with this intensity [21].

The number of sampled code block vectors depends on the number of code blocks in a parallel region and it grows exponentially with the number of cores. Thus, for highly parallel regions containing

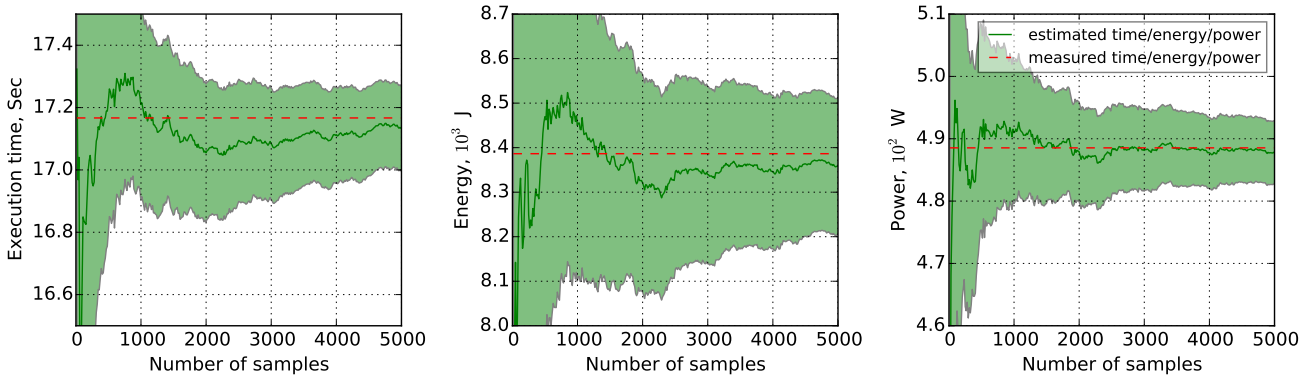

Figure 5: Estimated vs. measured execution time, energy and CPM power for a code block of `heartwall`(Sandy Bridge)

multiple code blocks, the number of code block vectors could render the analysis of profiling results slow, since we should account for the energy consumption of all vectors. In practice and in all applications with which we have experimented, strong temporal locality of executing code blocks keeps the number of vectors of code blocks that execute concurrently (and, thus, are sampled) low. For example, in `heartwall`, three code block vectors dominate energy consumption (Figure 4), while the total number of sampled code block vectors is about 100 in this benchmark.

3.5 Analysis of Model Accuracy and Convergence

ALEA provides confidence intervals for its time, power, and energy estimates. We have shown in earlier work [21] how to construct confidence intervals for the time and CPM power estimates.

Using the confidence intervals for execution time and power, we can also evaluate the worst case upper and lower bounds for energy consumption:

$$\hat{t}^{l}_{\overrightarrow{blm}} \cdot \widehat{pow}^{l}_{\overrightarrow{blm}} \leq e_{\overrightarrow{blm}} \leq \hat{t}^{u}_{\overrightarrow{blm}} \cdot \widehat{pow}^{u}_{\overrightarrow{blm}}. \tag{10}$$

where $\hat{t}^{l}_{\overrightarrow{blm}}$ ($\hat{t}^{u}_{\overrightarrow{blm}}$) is the lower (upper) bound of the confidence interval for the time estimates and $\widehat{pow}^{l}_{\overrightarrow{blm}}$ ($\widehat{pow}^{u}_{\overrightarrow{blm}}$) is the lower (upper) bound of the power confidence interval. We can increase the total number of samples to tighten the confidence interval for execution time. We can increase the number of samples of each code block to narrow the confidence interval for power estimates. Thus, to narrow both confidence intervals, we can increase the total number of samples and the number of samples per block.

To demonstrate the accuracy improvements of our time, energy and CPM power estimates as the number of samples increases, Figure 5 shows their evolution for a coarse-grained code block from the `heartwall` benchmark. We run this benchmark on our Sandy Bridge server (see Section 4.1). The dark green line represents the time (left), energy (middle) and power (right) estimates, whereas the lighter green area shows the 95% confidence interval for each estimate. The red dashed line in each plot corresponds to direct measurements for the block averaged over several runs.

The minimum number of samples for the block, collected during the first run of the experiment, is 16. The error of the energy estimate from the 16 samples, compared to direct measurement, is 12%. This error stabilizes below 1.7% with 110 samples and following that drops slowly and becomes lower than 1.5% after 421 samples and lower than 1.2% after 2300 samples. Other benchmarks exhibit similar behavior.

In the case of VPM, the width of the confidence interval for power estimates depends on the number of samples of the first instruction and the number of samples of the last instruction. VPM needs twice as many samples to achieve the accuracy of CPM.

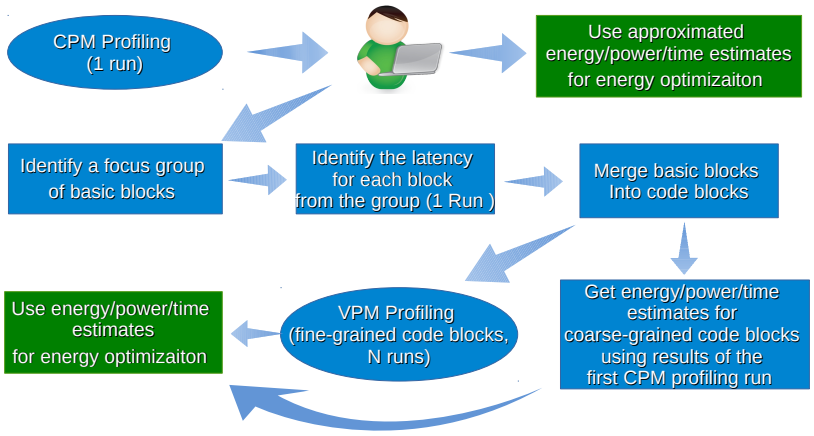

Figure 6: The ALEA profiling process

We could reduce the length of the sampling interval to obtain more samples during each profiling run. However, sampling uses interrupts that distort program execution, thus introducing errors in the time and energy estimates. We adopt a less intrusive but costlier solution that performs more profiling runs and combines their samples into a single set. This approach improves the accuracy of our energy estimates and also increases the probability of sampling fine-grained code blocks. In practice, each run of an application is unique and the total execution time of the application changes slightly from run to run, even though the application is executed under the same conditions. To address this issue, we average the total execution time over the runs.

4 Implementation

ALEA uses an online module that runs as a separate control process. This module samples the application's instruction pointer and power levels. It collects profiling information at random points transparently to the application. The sampling process has four stages. It first measures power and then attaches to the program threads through `ptrace`. Next, it retrieves the current instruction pointer, and finally detaches from the threads. The control process sleeps between samples to minimize application impact.

The online module transfers the results of profiling, including sampled instruction addresses and power measurements, to an offline module that derives energy estimates. The offline module uses `objdump` to disassemble the profiled program, and its own parsers to identify basic blocks and instructions inside these blocks. It uses this information to assign samples to specific blocks. Thus, ALEA can estimate the execution time for each basic block, using the probabilistic model described earlier. To enable accurate energy and power profiling, the user should identify basic blocks to be merged into coarse-grained blocks if the default power model (CPM) is used. VPM can be applied to fine-grained blocks (with latency as low as 10μs, see Section 5.3), however the user should point ALEA to these blocks. The user obtains the latency of each basic block by using an additional run and dynamic instrumentation tools, or direct instrumentation. Finally, we use DWARF debugging information to associate ALEA's estimates with specific lines in the source code.

Figure 6 illustrates the process of profiling from the user perspective. During the first run of ALEA, the user identifies execution time, energy and power estimates for each basic block using CPM. From this point the user can do additional runs and select basic blocks where more accurate estimates are needed.

This step can also be automated with hotspot analysis, using, for example, clustering and classification techniques similar to the ones used in architectural simulators [23]. The user should point ALEA which basic blocks should be merged into code blocks using addresses of these blocks. If latency of a merged code block is greater than the power-sensing interval then the results of the first profiling run could be used to get accurate power and energy estimates for this code block. If the latency of the code block is less than the power-sensing interval and no lower than 10 μs, then the user needs to run VPM for the specific code block and pass to ALEA the addresses of the first and last instructions and the latency of the block. The user can initiate any desirable number of profiling runs using CPM or VPM to achieve the desirable accuracy.

We note that ALEA can be implemented in the kernel space, which would reduce its overhead and allow sampling at a higher frequency than a user-level implementation. Nonetheless, we opted for a user-level implementation to enable easier deployment of the tool in environments where the user can not have access to the kernel.

4.1 Platforms and Energy Measurement

ALEA builds on platform-specific capabilities to measure power. It can use direct power instrumentation, integrated power sensors, or software-defined power models. ALEA operates under the sampling rate constraints of the underlying sensors, but is not limited by them since its probabilistic models can estimate power at finer code block granularity. To demonstrate the independence of our approach from the specifics of the hardware, we use two platforms that differ in architectural characteristics and support for power instrumentation.

Our first platform, a Xeon Sandy Bridge server, uses RAPL for on-chip software-defined modeling of energy consumption, which we obtain by accessing the RAPL MSRs [19]. The server includes 4 Intel Xeon E7-4860 v2 processors, each with 12 physical cores (24 logical cores) per processor, 32 KB I and D caches per core, a 3 MB shared L2 cache, and a 30 MB shared L3 cache per package. The system runs CentOS 6.5 and has a maximum frequency of 2.6 GHz.

Our second platform, an ODROID-XU+E board [24], has a single Exynos 5 Octa processor based on the ARM Big.LITTLE architecture, with four Cortex-A15 cores and four Cortex-A7 cores. In our experiments we only use the Cortex-A15 cores at their maximum frequency of 1.6 GHz. Each core has 32 KB I and D-caches, while the big cores also share a 2 MB L2 cache. We measure power through the board's integrated sensors. The system runs Ubuntu 14.04 LTS.

4.2 Power-Sensing Interval

Our Intel Sandy Bridge server provides energy estimates through RAPL, but not power measurements. We measure processor power consumption on it by dividing energy measurements taken at the start and end of the power-sensing interval by the size of this interval. The energy counter is updated only once per millisecond, which is the server's minimum feasible power-sensing interval. Our Exynos platform includes the TI INA231 power meters [25], which directly sample power consumption for the whole system-on-chip and several of its components. These samples are averaged over a user-defined interval. The minimum available interval on the Exynos is 280 μs.

Shorter power-sensing intervals improve CPM estimates for the fine-grained blocks. Thus, our CPM experiments use the minimum power-sensing interval on both platforms.

VPM assumes that we can precisely control when power is measured. Unfortunately, on Intel processors, RAPL energy counters are updated by the hardware at unknown points and transparently to software, with an update frequency of one ms. Thus, we cannot use VPM for Intel processors with RAPL. On the Exynos platform, even though we initiate the power measurement, a small variable delay occurs between the system call to read the power counters and the actual reading. Still, we can assume

Figure 7: Overhead per sample versus number of threads with ALEA running on a dedicated or a shared core.

that this delay remains fixed on average and converges to its mean value as the number of samples increases.

In theory, the granularity of the power meters does not limit VPM so the power-sensing interval is not critical. In practice, our experiments show that the maximum overall latency of the first stage of sampling in VPM is about 650 μs. At the first stage ALEA calls the interface that starts power measurements and the attach interface that stops all threads. At the second stage the tool calls the interface that takes the power reading, delivers the instruction pointer for each thread and then detaches from all threads. While in CPM the attach interface is called after the power measurement, in VPM it is part of the measurement. For our power reading to contain any information about the profiled code block, the power sensing interval must be longer than the first stage sampling latency. For this reason we set the power-sensing interval to 1100 μs.

4.3 Sampling Interval and Overhead

The attach, deliver, and detach phases of the tool introduce overhead. The attach phase interrupts the execution of all threads. Threads continue to execute after completion of the detach phase.

We use a set of benchmarks from the NAS suite (`BT`, `CG`, `IS` and `MG`) to illustrate the profiling overhead of the ALEA online module, with a core dedicated to each application thread. We choose benchmarks that scale well so we can evaluate the overhead of ALEA under a difficult scenario in which the application utilizes the processor resources fully and is sensitive to sharing them with ALEA.

Figure 7 shows the overhead of ALEA, which clearly depends on the number of application threads, since the tool has to deliver the instruction pointer for each thread. With more threads the delivery overhead necessarily increases. The profiler can share a core with the program or it can use a dedicated core. We observe higher overhead when the tool and the program share a core, which is caused by a delay in the delivery phase. This delay grows significantly, up to 10×, when the tool attaches to the application thread on the shared core due to the overhead of switching between an application thread and the ALEA profiler. The profiler leverages the OS load-balancing mechanisms to exploit idle cores, if available. The benchmarks occasionally leave cores idle and this slack grows as the benchmark uses more threads. Linux dynamically migrates profiled threads from a core used by ALEA to an idle one to eliminate the context-switching overhead, which leads to an overhead reduction as the number of threads grows on the Sandy Bridge platform when ALEA uses a shared core. Still, if the benchmarks use more than 30 threads the context-switching overhead becomes negligible in comparison to the overhead caused by the attach, deliver, and detach phases.

These findings suggest that we should choose the sampling interval based on the number of threads and the availability of a dedicated core in order to reduce the bias in the estimates. However, for most programs the thread count is not constant, so the overhead and the optimal sampling interval vary during execution. We cannot change the sampling interval dynamically since our probabilistic model assumes a uniform sampling process. Instead, the sampling interval should reflect the average overhead

measured over a whole run of the application. ALEA reports the average overhead for each profiling run by measuring the runtime of the sampling phases. Thus, the user can easily find a sampling interval with an acceptable overhead. By default, ALEA uses a sampling interval that keeps overhead under 1% when all available cores are used by an application. It also estimates the optimal sampling interval that gives the minimum overhead and the maximum number of samples after a profiling run.

5 Validation

In this section we validate the accuracy of our approach under extreme scenarios, where the power consumption changes rapidly and widely. We use multiple benchmarks on both hardware platforms to establish the generality of our results. We validate ALEA using loops with coarse-grained blocks for each platform. We then show that CPM (Section 5.1) can accurately estimate the energy consumption of coarse-grained blocks, while VPM (Section 5.2) does so even for fine-grained blocks.

5.1 Constant Power Model Validation

To validate the accuracy of ALEA's energy consumption estimates, we use ten parallel benchmarks from the NAS and Rodinia suites. We use a wide range of benchmarks to achieve good coverage of loop features in terms of execution time and energy consumption, including loops with distinct power profiles and power variation between their samples. For each benchmark we compare CPM time and energy estimates for each loop with direct measurements. These direct measurements use instrumentation at the entry and exit points of the loops. On the Sandy Bridge platform we instrument the code using the `rdtsc` instruction and the RAPL interface for time and energy respectively. On Exynos we use `clock_gettime()` to measure time and the `ioctl()` interfaces to access the INA231 power sensors.

We compile the Rodinia benchmarks using the flags specified by the benchmark suite and run them with the default inputs. The benchmarks from NAS are built using option `CLASS=B` and run with the default options. We use the abbreviation `suite.benchmark` to indicate the benchmarks on our graphs. For example, `R.sc` corresponds to the StreamCluster (SC) benchmark from the Rodinia suite.

5.1.1 Sandy Bridge

We run each benchmark using 96 threads on the Sandy Bridge platform. ALEA shares a core with an application thread. We use a three second sampling interval to achieve a runtime overhead of under 1%. The average latency of most instrumented loops is less than this sampling interval.

We combine samples from multiple runs for each benchmark to achieve accurate estimates. We collected 500 runs for each benchmark but found the accuracy with 180 to be sufficient. Using more runs has little effect, so our remaining experiments use 180 runs.

Figure 8 shows how the error of energy estimates changes with the number of combined runs. Ignoring `StreamCluster`, the mean error is as high as 32% when we profile each benchmark only once. Predicting energy for `StreamCluster` requires more than one run. The total execution time of one instrumented loop is shorter than the sampling interval, so the loop may not be sampled during a run.

Increasing the number of combined runs reduces the error. On average, the error decreases to 2.1% after 180 profiling runs. Even for `StreamCluster` the error is only slightly above average at 2.3%. Regarding the estimation of the total energy consumption of the program, the average error across all benchmarks is 1.6% for 180 runs. The energy estimates errors of ALEA are thus comparable to ALEA's runtime overhead. For completeness, we also profile a 94-thread version of each benchmark with ALEA using a dedicated core. We used a sampling interval of 3 seconds to keep overhead low (under 1%). The minimum error of 1.9% with 155 runs is again comparable to the runtime overhead.

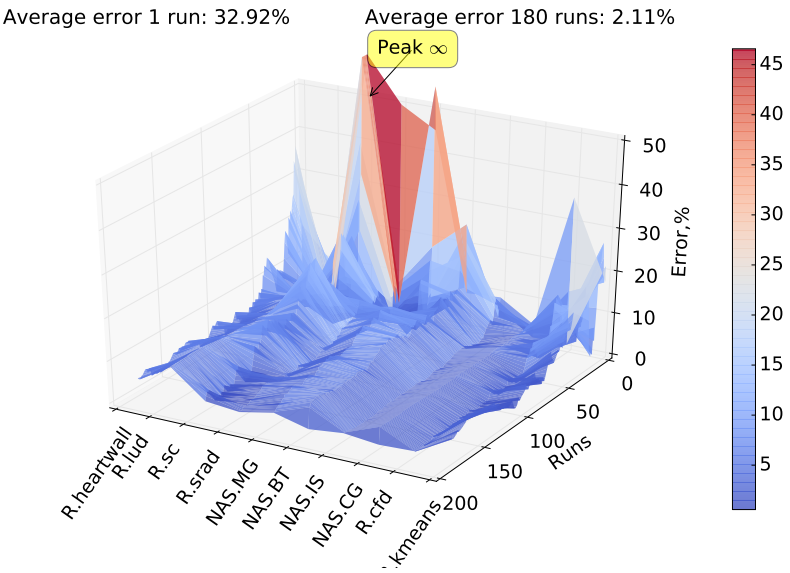

Figure 8: Average error of energy estimates across all loops on Sandy Bridge

5.1.2 Exynos

We similarly validate ALEA on the Exynos platform. We used the same experimental setup, except that we use two threads for the benchmarks to give ALEA a dedicated core and set the sampling interval to 500 ms with the aim of keeping the overhead slightly below 1%. Figure 9 presents the trade-off between the number of runs and the average error in our energy estimates for the loops. The sampling interval is longer than the average latency of almost all instrumented loops. The average error of the energy estimates is about 21% for a single profiling run but decreases to 1% with 50 or more profiling runs. The whole program absolute error for energy, averaged across all benchmarks, is 1.1% for 50 profiling runs.

ALEA exhibits its highest error in the energy estimates of `heartwall` from the Rodinia suite. A loop at line 57 in avimod.c, has a total execution time of 0.31 seconds, less than the sampling interval. Thus, it was sampled only once during the first profiling run and the error in terms of both execution time and energy exceeded 120%. However, this error falls below 5% within 50 runs. While ALEA can theoretically achieve any desirable accuracy, in practice the lower limit of the error is about 1% due to the instrumentation overhead and measurement interference.

We repeat our experiments using 4 threads for each benchmark, with ALEA sharing a core. The results show the same patterns, with the average error of the energy estimates at about 1.3% and stabilizing after 110 profiling runs.

5.2 Variation-Aware Power Model Validation

Next we validate the estimates produced by our Variation-Aware Power Model (VPM). These experiments only use the Exynos platform due to RAPL's limitations (see Section 4.2). We design VPM to capture fine-grained program power behavior. Thus, we validate it on programs with significant power variation between fine-grained blocks: loops with latency less than the power-sensing interval. We use a 1100 μs power-sensing interval for the reasons discussed in Section 4.2. On the other hand, to validate VPM we must use loops with known power consumption and latency. To measure them directly, latency must be above the minimum power-sensing interval (280 μs).

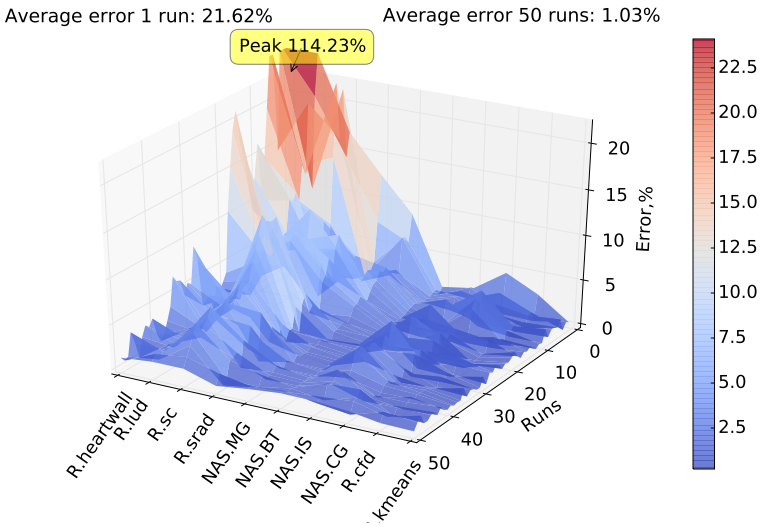

Figure 9: Average error of energy estimates across all loops on Exynos

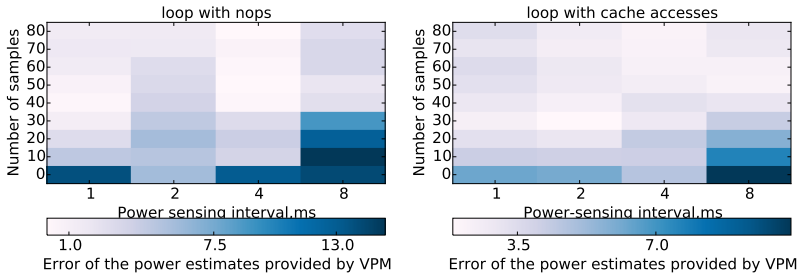

Figure 10: Power estimates provided by VPM for the microbenchmark

5.2.1 Context-driven power variation

Our experiments show that processor dynamic power consumption is primarily affected by cache and memory access intensity [21]. Thus, we use a microbenchmark that consists of two loops. The first loop contains a block with `nop` instructions, consuming 1.68 Watts. The second loop contains a block with memory access instructions engineered to hit in the L1 cache, and consumes 2.54 Watts. We control the latency of each loop through the number of iterations. We set the latency of each loop close to 400 μs, which is within the range of latencies discussed above.

We profile a 2-thread version of the microbenchmark 1000 times (ALEA runs on a dedicated core) to collect at least 80 samples, which ensures at least a 95% confidence interval of the power measurements within 1% of the mean. Figure 10 shows the trade-off between the number of samples and the average error of the power estimates provided by the VPM model, which we compare to direct power measurements. In this figure the darkness of the color indicates the average error of the power estimates. We change the power-sensing interval from 1 ms to 8 ms and use a one second sampling interval to keep overhead under 1%.

Figure 10 demonstrates that the average error of VPM power estimates decreases with the number of samples. 40 samples are enough to estimate the power consumption of the loops with an error less than 4%, even if we use an 8 ms power-sensing interval, which is 20$\times$ more than the latency of these loops. The average error of VPM power estimates is about 3.5% for each power-sensing interval when the number of samples is close to 80 ($\sim$3.5% for the energy estimates).

Even though VPM provides precise power estimates, interference from power measurements im-

pacts its accuracy. While we expect the average error to be higher for a 4 ms interval than a 2 ms interval, our results show the opposite. We speculate that this happens because of OS interference, but performing kernel modifications to validate this hypothesis was beyond the scope of this paper.

We use the same experiment and 1000 runs to test CPM's accuracy. Similarly to VPM, we vary the power-sensing interval from 1 ms to 8 ms. Regardless of the interval, the average error of CPM power estimates is 45.5% after 1000 runs (∼45.5% for the energy estimates), which is 12× higher than the average error of VPM. Clearly, VPM can successfully handle power variations across fine-grained blocks, which CPM and other known power measurement and modeling techniques can not.

5.2.2 Concurrency-driven power variation

We next investigate how CPM and VPM handle power variations due to switching from executing parallel coarse-grained loops to executing sequential fine-grained loops. The sequential loops consume less power since fewer cores are active. The following benchmarks have sequential fine-grained loops that are surrounded by parallel coarse-grained loops: `Srad (Rodinia)` and `StreamCluster (Rodinia)`. For those fine-grained loops, we artificially inflate their execution time, to allow direct measurements.

Similarly to the microbenchmark experiments, we run each benchmark 1000 times. We use 2 threads for each benchmark and run ALEA on a dedicated core. Similarly to the context-driven power variation experiment, we use a one second sampling interval to keep the overhead negligible. Our experiments show that the average error of VPM power estimates is 5.4% (∼5.5% for energy estimates), while the average error of CPM estimates is 15.7% (∼16% for energy estimates), almost 3× higher.

We repeat the experiments with 4 application threads (so ALEA must share a core) and a 1 ms power-sensing interval. Our results demonstrate that the average error of VPM power estimates is 6% (∼6% for energy estimates), while the average error of CPM estimates is 25% (∼25% for energy estimates). Again, VPM can handle programs with fast changing power behavior better than CPM and produces accurate estimates.

5.3 Limitations of ALEA

ALEA is a tool for fine-grained energy profiling. The probabilistic model of the tool makes it possible to estimate the execution time of a code block with high accuracy even if the average execution time of the block is a few microseconds, while the sampling interval can be several seconds. However, our base power model (CPM) assumes that power remains constant during the power-sensing interval. VPM relaxes this assumption and captures power variation over intervals that are shorter than the power-sensing interval. Nonetheless, the resolution of this model is also limited.

To explore how VPM captures power variation over an interval with a latency of a few microseconds or nanoseconds, we employ a microbenchmark similar to the one with which we validated VPM. The microbenchmark contains two loops: the first loop executes only `nop` instructions and in each iteration of the second loop we access the same 368 bytes fetched into the L1 cache after their initialization. We test the resolution limit of VPM by progressively reducing the execution time of the loops. We control loop execution time by changing the number of iterations. We ensure that each iteration executes the same instructions and the same number of L1 cache accesses in particular. Thus, loop power consumption should remain constant when we reduce the iteration count, but the difficulty of estimating it accurately will progressively increase. We do not use the parallel version of the microbenchmark since the system calls starting and synchronizing threads inside the parallel regions already take hundreds of microseconds.

Figure 11 shows the results for the microbenchmark with loop latencies between 1 and 400 μs. Dashed red lines correspond to the power consumption of each loop found in the experiments in Section 5.2.1. The leftmost plot depicts how the power estimates for the loops change with their latency. The middle

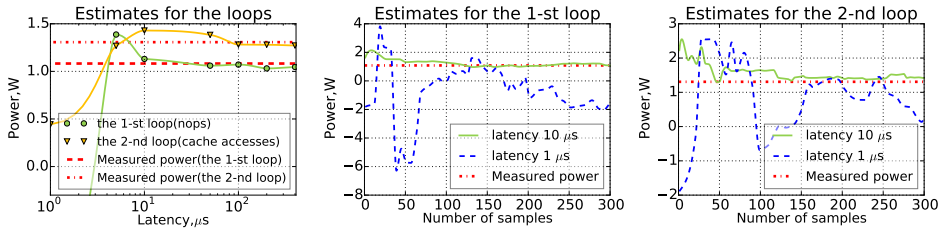

Figure 11: Power estimates provided by VPM for the microbenchmark with reduced latencies of the loops

plot shows how the power estimates for the loop with nops change with the number of samples, while the rightmost plot demonstrates the same for the loop with cache accesses. While we cannot directly validate the power consumption since the loop latencies are less than the minimum power-sensing interval (280 μs), we expect that the difference in power consumption between these loops should remain constant for all loop latencies.

We observe the expected difference for latencies of at least 10μs (leftmost subplot of Figure 11). While ALEA can theoretically achieve any desirable accuracy, in practice the lower limit of the error is about 1% due to the instrumentation overhead and measurement interference. The error of the power estimates for 10μs is 6.9%. This latency is the maximum resolution of the power estimates provided by VPM. On the other two subplots, we also observe that the power estimates for the loops with this latency clearly converge to the measured power as the number of samples grows. For latencies below 10μs, the estimates never converge due to the limited accuracy of the power sensors. To capture power variations over intervals with 1μs latency, the sensors should provide accuracy of measurements within 10^{-4} Watts if a 1100 μs power-sensing interval is used. Moreover, the time required to measure power varies over samples and this variation exceeds 1μs, which negatively impacts the power estimates.

ALEA could not reliably capture power variation between code blocks with execution times less than 10μs. Nonetheless, this interval is still 110 times shorter than the used power-sensing interval and 28 times shorter than the minimum feasible power-sensing interval in our platforms. As we show later, this resolution is sufficient to support energy optimization. To the best of our knowledge, ALEA is the first tool that provides accurate power estimates at this resolution.

6 Discussion

6.1 Hysteresis in Power Measurements

Physical power measurements are affected by bypass capacitance, inductance and impedance components [26, 27]. A voltage regulator module delivers power to a processor through a Power Distribution Network (PDN). This network contains impedance and inductance components located at different levels: motherboard, package and die. By executing different code blocks, a processor produces variations in current demand. Due to current spikes and impedance components, a voltage drop can occur across the network [28]. In turn, the inductance components introduce voltage fluctuations because of the `di/dt` effect [29, 30, 31]. Thus, voltage variations occur.

The voltage regulator module adjusts the voltage level to mitigate these variations. However, it cannot always change voltage quickly enough. To maintain voltage at safe levels when the current increases significantly, the PDN uses decoupling capacitors close to the processor (or package or motherboard). On the Exynos board the power sensors (INA231) take power measurements on a power rail that connects the PMIC (Power Management Integrated Circuits, `MAX77802`), which provide a voltage regulator, with the chip. Decoupling capacitors, inductance and impedance components located between the power

sensors and the die affect instantaneous power dissipation. Thus, the effect of a code block on the measured power is not instantaneous and hysteresis occurs between execution of the block and its effect on the measured power consumption. Moreover, capacitance components also smooth out instantaneous power dissipation.

ALEA's probabilistic sampling can theoretically capture power shifts during instruction execution at any granularity. In practice, several factors, including software overheads and the overheads and measurement error of the power sensing instruments, prevent such very fine granularity. The length of the hysteresis, i.e. the typical response time of the PDN, is orders of magnitude less than the minimum code block duration for which ALEA can probabilistically estimate power. The literature reports PDN response times in the range of 75 nanoseconds [32], versus ALEA's 10 μs minimum code block length and minimum power-sensing intervals on the order of hundreds of μs on the Exynos and Intel platforms.

6.2 Power modeling with performance counters

One of the possible solutions to break the coarse granularity of power sensors and mitigate hysteresis effects is to predict power consumption from hardware activity vectors. Previous research has extensively investigated the use of regression models on activity vectors of hardware performance counters to estimate power consumption. These models assume that processor power consumption correlates with the frequency of certain performance impacting events. Performance counters can be accessed with low overhead on most modern processors, which makes them appropriate for fine-grained power modeling. We modify ALEA to use a regression model for power modeling following Shen et al [33]. We use PAPI to measure non-halt core cycles, retired instructions per CPU cycle, floating point operations per cycle, last-level cache requests per cycle, and memory transactions per cycle over the power-sensing interval and then correlate these measurements to the sampled power consumption. We build the regression model for each platform individually on the basis of ten benchmarks used for validation of CPM and VPM with varying thread counts. We use a sampling interval that keeps runtime overhead under 1%. We add L1 cache accesses to the model because its power consumption is highly sensitive to this event and omitting it loses significant accuracy.

In this experiment we use linear regression to predict power consumption for samples that we use for training and compare the results of prediction with the power measurements taken for these samples. The average error of the power estimates provided by the model is about 10% on both platforms, which is acceptable and consistent with the results demonstrated by Shen et al. However, the regression models produce anomalies by predicting the power of cache access intensive code blocks to be lower than the power of code blocks with `nop` instructions. Thus, accuracy of the modeling could be insufficient to capture the effect of code blocks on power consumption, even though the average error of the linear regression is relatively small.

Intel's RAPL provides access to a digital power meter which predicts power consumption on the basis of performance event counters, similar to linear regression models [19]. However, RAPL uses about 100 internal micro-architecture counters [34]. Unfortunately, no more than 12 performance counters can be accessed by system software at once through the Performance Monitoring Unit (PMU). On the Exynos platform, the PMU module allows the user to sample only 7 performance counters at once. Thus, it is likely that RAPL achieves better accuracy by having fast hardware-supported access to many more performance counters than our model.

7 Use Cases

We now present practical examples of how ALEA enables energy-aware optimizations We first explore the effect of energy optimizations on power consumption in a latency-critical financial risk application where we apply VPM. We then demonstrate results for energy optimization based on runtime DVFS control using CPM.

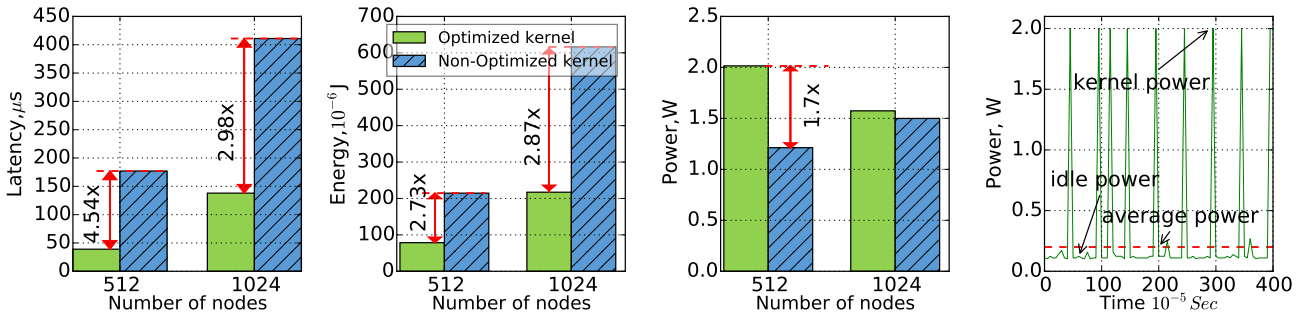

Figure 12: Latency, energy and power for the binomial option pricing kernel. The rightmost plot presents power changes during run of the workload for the optimized kernel(512 nodes)

7.1 Energy optimization of a latency-critical workload

We applied ALEA to profile and optimize a real-time and latency-critical option pricing workload. An option pricing model allows financial institutions to predict a future payoff using the current price and volatility of an underlying instrument. However, the current price is valid only for a limited interval, which implies that a decision about buying or selling the instrument is time critical. Binomial option pricing is a popular pricing model that enables accurate prediction with low computational requirements [35]. We use a commercial binomial option pricing kernel and a realistic setup that delivers requests to a worker thread running the kernel. The setup consists of two threads: the first thread listens to a TCP/IP socket, receives messages and fills a queue with the messages; the second thread parses the messages and transfers requests to the worker thread. A third thread, the worker thread, processes option pricing requests.

The option pricing kernel uses the current price, volatility and number of nodes in the binomial tree as input parameters. The latency of the kernel depends only on the number of nodes since variation of the current price or volatility does not affect control and data flows of the kernel. Accuracy of the prediction is also primarily affected by the number of nodes in the binomial tree. While the current price and volatility could change in option pricing requests, in practice financial institutions use the minimal number of nodes which guarantees an acceptable quality of the prediction to reduce a pricing kernel response time. Consistent with this, the latency of the kernel should be a constant if the number of nodes does not vary. Following private communication with a leading financial institution, we used 512 nodes for realistic runs. The latency of the binomial option pricing kernel, which is about 177 μs, is less than the power-sensing interval (280 μs) for this number of nodes on the Exynos platform.

According to the results of energy profiling, two code blocks of the kernel with co-running system threads consume more than 98% of the energy spent per request: the first code block builds a binomial tree and the second block computes the value of the option used to estimate the future payoff. To build the binomial tree, the first code block calls the exponential function in a loop, which is expensive in terms of energy and performance. The parameters of the function depend on an iteration of the loop and the underlying volatility, which is constant per request. To improve energy efficiency, we modified the kernel to dynamically identify the most frequently occurring volatility in option pricing requests and build at runtime an array containing the results of the exponent call for each iteration and this volatility (an example of memoization). In the optimized version of the kernel, we use the precomputed results instead of calls of the exponential function when the volatility of a request conforms with the most frequently occurring volatility.

Our experiments show that in 99.9 % of the samples, the first system thread executes the `do_futex_wait` system call and the second thread executes the `epoll_wait` system call when the worker thread runs the kernel. Both system calls keep a core in sleep mode waiting for an incoming network message

(the first thread) or a message from the queue (the second thread). The highest power consumption is observed when the worker thread runs the kernel.

We apply VPM to profile the kernel while reproducing a real workload of option pricing requests from an actual trading day in the NYSE. Figure 12 shows latency, energy and power consumption of the optimized and non-optimized versions of the kernel. By applying memoization, we managed to reduce latency of the kernel by 2.98 times and energy consumption by 2.87 times with binomial trees of 1024 nodes. We also reduced latency by 4.54 times for trees with 512 nodes. However, the energy reduction in this case was only 2.73 times. We explain this by a power spike that accompanied the latency reduction: the difference in the power consumption between the optimized and non-optimized versions for 512 nodes is 1.7 times, while this difference is only 4% for 1024 nodes. The explanation for the spike is that the number of cache accesses executed per second is higher in the optimized kernel with 512 nodes. These results demonstrate that energy and performance optimizations could lead to a significant rise in power consumption – which is a critical parameter for datacenter operators. ALEA provides the user with an option to choose an acceptable level of performance and energy optimization that meets a power budget. For example, on our Exynos platform, the user should not use the optimized version for binomial tress with 512 nodes and power capped at 2 Watt. As follows, ALEA enables users to optimize code under controlled power consumed at the fine-grained level without applying more intrusive power capping techniques such as DVFS [36].

ALEA reveals a further interesting finding. The average power consumption over the entire workload run with the optimized kernel (512 nodes) is 0.2 Watt (rightmost plot of Figure 12); this is 10 times less than the power consumption of the kernel. The difference is due to power variations during the workload run: the minimum power consumption (0.12 Watt) is observed when all threads stay idle waiting for incoming requests and the maximum power consumption (2 Watt) is observed when the kernel is running. Furthermore, the average power consumption over the entire run with the non-optimized version is 0.28 Watt, which is 40% higher than the average power consumption of the workload for the optimized version (0.2 Watt). In contrast, the power consumption of the optimized kernel is 1.7 times higher than the power consumption of the non-optimized kernel. This observation suggests that the average power consumption of the entire workload cannot characterize the power consumption of the key kernel in the workload.

7.2 Energy optimization with runtime DVFS control

The frequency that minimizes energy consumption depends on how intensely the cores interact with parts of the system not affected by frequency scaling, especially the main memory. Most existing approaches control frequency at the phase granularity, based on performance profiling, performance counter information, or direct energy measurements. We show how to use ALEA to drive decisions at any DVFS-supported granularity.

We use the Needleman-Wunsch benchmark from the Rodinia suite with a 40000×40000 matrix (∼6GB) as its input, executed on the Sandy Bridge platform. We first identify the optimal frequency and amount of parallelism for the whole program by taking energy measurements for the entire program. We test all combinations of frequency and number of threads. The leftmost plot of Figure 13 shows the results. The x-axis is the number of threads (1 to 96), the y-axis is frequency (1.2 to 2.6+ GHz), while the darkness of the color indicates energy consumption. The highest point on the y-axis, 2.6+ GHz, corresponds to enabled Intel Turbo Boost, in which case the processor can automatically raise the frequency above 2.6 GHz. The minimum energy consumption (2801 Joules) is observed when using 21 threads at 2.2 GHz.

ALEA finds two distinct code blocks that consume 99% of the total energy. The first code block uses one thread to initialize the arrays used by the benchmark. The second code block processes these arrays in parallel. We profile these blocks using CPM at different frequencies and with different thread counts for the second block. The middle and the rightmost plots of Figure 13 show the profiling results for the second and first blocks respectively. The two blocks consume their minimum energy at different

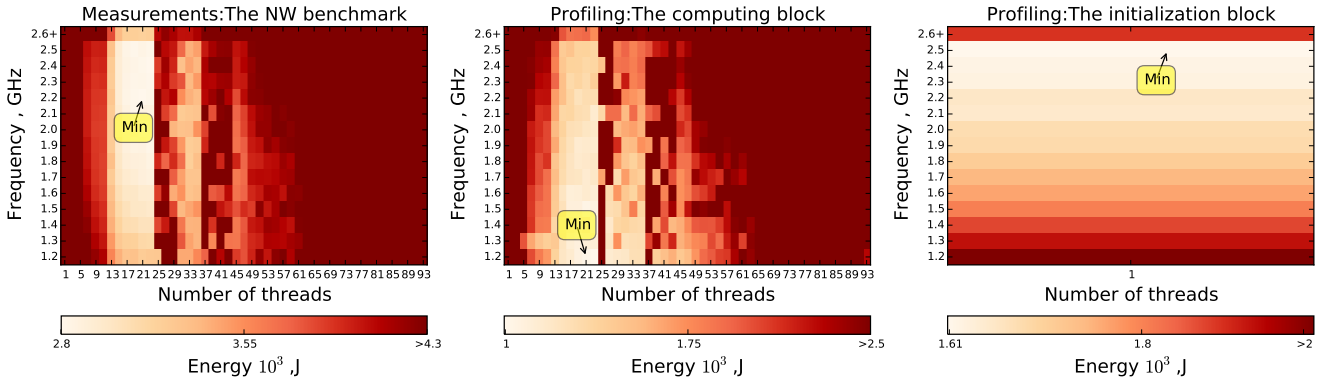

Figure 13: Energy consumption vs frequency and number of threads for the Needleman-Wunsch benchmark

frequencies, 2.5 GHz for the initialization block and 1.2 GHz for the computing block.

The blocks have different memory access patterns, which explains their energy profiles. Both read data from an array much larger than the caches with similar frequency. However, the initialization block has high temporal and spatial locality so most memory loads are serviced by the caches, processor stalls are few, and performance depends only on frequency. Thus, limiting the frequency too much increases execution time more than power reduction compensates. Alternatively, the computing block reuses data once or not at all, reading data from completely different pages in each iteration. Most accesses result in cache and `TLB` misses, stalling the processor and making it run effectively at the speed of main memory. Lowering the frequency of the processor has little effect on performance, so the optimal choice is to use the lowest frequency.

With this knowledge, we dynamically set the frequency to the optimal one at the beginning of each block. Compared to using a single frequency, optimal for the program as a whole, we reduced energy consumption by almost 9% (2581 Joules). In terms of peak power, which is reached when executing the parallel computing group, the improvement is significant, 41.6%, while performance degradation is only 3.5%.

Several works consider different approaches to reduce energy consumption through runtime DVFS [37]. However, most of these works use analytical models to predict the voltage and frequency setting which reduces energy consumption without a significant performance degradation. Central to such approaches is that the analytical models typically rely on parameters which have to be discovered for each platform individually. Moreover, the identified parameters do not guarantee the optimal voltage/frequency in terms of energy consumption. ALEA mitigates these restrictions by assigning accurate power and energy estimates to source code for each voltage/frequency set.

8 Related Work

PowerScope [2], an early energy profiling mechanism, profiles mobile systems through sampling of the system activity and power consumption, which it attributes to processes and procedures.

The tool does not use a probabilistic model and does not address randomness of the sampling process to enable fine-grained energy profiling. PowerScope assumes that for most samples the program executes only one procedure during the sampling interval. This assumption holds only for coarse-grained procedures with latency greater than the sampling interval, which is approximately 1.6 ms in PowerScope. Therefore, the tool cannot perform energy accounting at a fine-grained level. Furthermore, the accuracy of energy estimates is limited by the minimum feasible power-sensing interval provided by hardware sensors. In this paper we demonstrate that ALEA effectively mitigates all these restrictions by

following a probabilistic approach.

Several tools for energy profiling use manual instrumentation to collect samples of hardware event rates from hardware performance monitors (HPMs) [11, 7, 10, 38, 9, 17]. These tools empirically model power consumption as a function of activity rates. These rates attempt to capture the utilization and dynamic power consumption of specific hardware components. HPM-based tools and their models have guided several power-aware optimization methods in computing systems. However, they often estimate power with low accuracy. Further, they rely on architecture-specific training and calibration.

Eprof [16] models hardware components as finite state machines with discrete power states and emulates their transitions to attribute energy use to system call executions. JouleUnit [15] correlates workload profiles with external power measurement to derive energy profiles across method calls. JouleMeter [4] uses post-execution event tracing to map measured energy consumption to threads or processes. While useful, these tools can only perform energy accounting of coarse-grained functions or system calls, which severely limits the scope of power-aware optimizations that can be applied at compile time or run time.

Other approaches measure power consumption of distinct components. PowerPack [3] uses manual code instrumentation and platform-specific hardware instrumentation for component-level power measurement to associate power samples with functions. NITOS [5] measures the energy consumption of mobile device components with a custom instrumentation device. Similarly, LEAP [6] measures energy consumption of networked sensors with custom instrumentation hardware. These tools profile power at the hardware component level, thus capturing the contributions of non-CPU components, such as memories, interconnects, storage and networking devices. ALEA complements these efforts. ALEA's sampling method can account for energy consumed by any hardware component between code blocks, while its statistical approach overcomes the limitations of coarse and variable power sampling frequency in system components.

Other energy profiling tools build instruction-level power models bottom-up from gate-level or design-time models to provide power profiles to simulators and hardware prototyping environments [14, 13]. These inherently static models fail to capture the variability in instruction-level power consumption due to the context in which instructions execute. Similarly, using microbenchmarks [12] to estimate the energy per instruction (EPI) or per code blocks based on their instruction mix does not capture the impact of the execution context.

9 Conclusion

We presented and evaluated ALEA, a tool for fine-grained energy profiling based on a novel probabilistic approach. We introduced two probabilistic energy accounting models that provide different levels of accuracy for power and energy estimates, depending on the targeted code granularity. We demonstrated that ALEA achieves highly accurate energy estimates with worst-case errors under 2% for coarse-grained code blocks and 6% for fine-grained ones.

ALEA overcomes the fundamental limitation of the low sampling frequency of power sensors. Thus, ALEA is more accurate than existing energy profiling tools, when profiling programs with fast changing power behavior. ALEA opens up previously unexploited opportunities for power-aware optimization in computing systems. Its portable user space module does not require modifications to the application or the OS for deployment. Using ALEA, we profiled two realistic applications and evaluate the effect of optimizations enabled uniquely by ALEA on their energy and power. Overall, ALEA supports simpler, more targeted application optimization.

Acknowledgement

This research has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through grant agreements EP/L000055/1 (ALEA), EP/L004232/1 (ENPOWER), EP/M01567X/1

(SANDeRs), EP/M015823/1, EP/M015793/1 (DIVIDEND) and EP/K017594/1 (GEMSCLAIM) and by the European Commission under the Seventh Framework Programme, grant agreement FP7-610509 (NanoStreams) and FP7-323872 (SCORPIO). We are grateful to Jose R. Herrero (UPC), Pavel Ilyin (Soft Machines) and the anonymous reviewers for their insightful comments and feedback.

References

[1] Chang F, Farkas KI, Ranganathan P. Energy-Driven Statistical Sampling: Detecting Software Hotspots. *Proceedings of the 2nd International Conference on Power-Aware Computer Systems*, PACS'02, Springer-Verlag: Berlin, Heidelberg, 2003; 110–129.

[2] Flinn J, Satyanarayanan M. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. *2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[3] Ge R, Feng X, Song S, Chang HC, Li D, Cameron KW. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems* 2010; **21**(5):658–671.

[4] Kansal A, Zhao F. Fine-Grained Energy Profiling for Power-Aware Application Design. *SIGMETRICS Perform. Eval. Rev.* Aug 2008; **36**(2):26–31.

[5] Keranidis S, Kazdaridis G, Passas V, Igoumenos G, Korakis T, Koutsopoulos I, Tassiulas L. NITOS Mobile Monitoring Solution: Realistic Energy Consumption Profiling of Mobile Devices. *Proceedings of the 5th International Conference on Future Energy Systems*, e-Energy '14, ACM: New York, NY, USA, 2014; 219–220.

[6] McIntire D, Stathopoulos T, Kaiser W. Etop: Sensor Network Application Energy Profiling on the LEAP2 Platform. *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, ACM: New York, NY, USA, 2007; 576–577.

[7] Bertran R, Gonzalez Tallada M, Martorell X, Navarro N, Ayguade E. A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs. *IEEE Trans. Comput.* Jul 2013; **62**(7):1289–1302.

[8] Manousakis I, Zakkak FS, Pratikakis P, Nikolopoulos DS. TProf: An Energy Profiler for Task-Parallel Programs. *Sustainable Computing: Informatics and Systems* 2014; doi:http://dx.doi.org/10.1016/j.suscom.2014.07.004.

[9] Contreras G, Martonosi M. Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events. *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, ISLPED '05, ACM: New York, NY, USA, 2005; 221–226.

[10] Isci C, Martonosi M. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, IEEE Computer Society: Washington, DC, USA, 2003; 93–.

[11] Curtis-Maury M, Shah A, Blagojevic F, Nikolopoulos DS, de Supinski BR, Schulz M. Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores. *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, ACM: New York, NY, USA, 2008; 250–259.

[12] Shao YS, Brooks D. Energy Characterization and Instruction-Level Energy Model of Intel's Xeon Phi Processor. *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, IEEE Press: Piscataway, NJ, USA, 2013; 389–394.

[13] Tsoi KH, Luk W. Power Profiling and Optimization for Heterogeneous Multi-core Systems. *SIGARCH Comput. Archit. News* Dec 2011; **39**(4):8–13.

[14] Tu CH, Hsu HH, Chen JH, Chen CH, Hung SH. Performance and Power Profiling for Emulated Android Systems. *ACM Trans. Des. Autom. Electron. Syst.* Mar 2014; **19**(2):10:1–10:25.

[15] Wilke C, Götz S, Richly S. JouleUnit: A Generic Framework for Software Energy Profiling and Testing. *Proceedings of the 2013 Workshop on Green in/by Software Engineering*, GIBSE '13, ACM: New York, NY, USA, 2013; 9–14.

[16] Pathak A, Hu YC, Zhang M. Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof. *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, ACM: New York, NY, USA, 2012; 29–42.

[17] Schubert S, Kostic D, Zwaenepoel W, Shin KG. Profiling Software for Energy Consumption. *2012 IEEE International Conference on Green Computing and Communications* 2012; **0**:515–522.

[18] Brouwers N, Zuniga M, Langendoen K. NEAT: A Novel Energy Analysis Toolkit for Free-Roaming Smartphones. *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, ACM: New York, NY, USA, 2014; 16–30.

[19] Rotem E, Naveh A, Ananthakrishnan A, Weissmann E, Rajwan D. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro* March 2012; **32**(2):20–27, doi:10.1109/MM.2012.12.

[20] Cao T, Blackburn SM, Gao T, McKinley KS. The Yin and Yang of Power and Performance for Asymmetric Hardware and Managed Software. *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, IEEE Computer Society: Washington, DC, USA, 2012; 225–236.

[21] Mukhanov L, Nikolopoulos DS, Supinski BRD. Alea: Fine-grain energy profiling with basic block sampling. *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2015.

[22] Graham SL, Kessler PB, Mckusick MK. Gprof: A call graph execution profiler. *SIGPLAN Not.* Jun 1982; **17**(6):120–126, doi:10.1145/872726.806987. URL http://doi.acm.org/10.1145/872726.806987.

[23] Sherwood T, Perelman E, Calder B. Basic block distribution analysis to find periodic behavior and simulation points in applications. *Technical Report*, La Jolla, CA, USA 2001.

[24] Hardkernel. Odroid-xu+e 2016. URL http://www.webcitation.org/6f2nShdcN., accessed: 2016-02-04.

[25] TI. *High- or Low-Side Measurement, Bidirectional CURRENT/POWER MONITOR with 1.8-V I2CTM Interface* 2013.

[26] Gupta MS, Oatley JL, Joseph R, Wei GY, Brooks DM. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. *2007 Design, Automation Test in Europe Conference Exhibition*, 2007; 1–6, doi:10.1109/DATE.2007.364663.

[27] Smith LD, Anderson RE, Forehand DW, Pelc TJ, Roy T. Power distribution system design methodology and capacitor selection for modern cmos technology. *IEEE Transactions on Advanced Packaging* Aug 1999; **22**(3):284–291, doi:10.1109/6040.784476.

[28] Bertran R, Buyuktosunoglu A, Bose P, Slegel TJ, Salem G, Carey S, Rizzolo RF, Strach T. Voltage noise in multi-core processors: Empirical characterization and optimization opportunities. *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, IEEE Computer Society: Washington, DC, USA, 2014; 368–380, doi:10.1109/MICRO.2014.12. URL http://dx.doi.org/10.1109/MICRO.2014.12.

[29] Joseph R, Brooks D, Martonosi M. Control techniques to eliminate voltage emergencies in high performance processors. *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, 2003; 79–90, doi:10.1109/HPCA.2003.1183526.

[30] Zhang X, Tong T, Kanev S, Lee SK, Wei GY, Brooks D. Characterizing and evaluating voltage noise in multi-core near-threshold processors. *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, IEEE Press: Piscataway, NJ, USA, 2013; 82–87.

[31] Das S, Whatmough P, Bull D. Modeling and characterization of the system-level power delivery network for a dual-core arm cortex-a57 cluster in 28nm cmos. *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, 2015; 146–151, doi:10.1109/ISLPED.2015.7273505.

[32] Grochowski E, Ayers D, Tiwari V. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, 2002; 7–16, doi:10.1109/HPCA.2002.995694.

[33] Shen K, Shriraman A, Dwarkadas S, Zhang X, Chen Z. Power containers: An os facility for fine-grained power and energy management on multicore servers. *SIGPLAN Not.* Mar 2013; **48**(4):65–76, doi:10.1145/2499368.2451124. URL http://doi.acm.org/10.1145/2499368.2451124.

[34] Rotem E, Naveh A, Rajwan D, Ananthakrishnan A, Weissmann E. Power management architecture of the 2nd generation intel core microarchitecture, formerly codenamed sandy bridge 2011. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7477510, hot Chips.

[35] Georgakoudis G, Gillan C, Sayed A, Spence I, Faloon R, Nikolopoulos D. Methods and metrics for fair server assessment under real-time financial workloads. *Concurrency and Computation: Practice and Experience* 3 2016; **28**(3):916–928, doi:10.1002/cpe.3704.

[36] Petoumenos P, Mukhanov L, Wang Z, Leather H, Nikolopoulos DS. Power capping: What works, what does not. *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*, 2015; 525–534, doi:10.1109/ICPADS.2015.72.

[37] Wu Q, Martonosi M, Clark DW, Reddi VJ, Connors D, Wu Y, Lee J, Brooks D. A dynamic compilation framework for controlling microprocessor energy and performance. *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, IEEE Computer Society: Washington, DC, USA, 2005; 271–282, doi:10.1109/MICRO.2005.7. URL http://dx.doi.org/10.1109/MICRO.2005.7.

[38] Li D, de Supinski BR, Schulz M, Nikolopoulos DS, Cameron KW. Strategies for Energy-Efficient Resource Management of Hybrid Programming Models. *IEEE Trans. Parallel Distrib. Syst.* Jan 2013; **24**(1):144–157.